SALT

A 2025 Perspective on Protecting Against the OWASP API Security Top 10 with Salt Security.

api

Fearless API Security

A 2025 Perspective on Protecting Against the OWASP API Security Top 10 with Salt Security

Introduction

Although the OWASP API Security Top 10 was published in 2023, its outlined threats remain very relevant. This whitepaper revisits these top threats, offering insights on their importance in 2025 and explaining how Salt Security can effectively defend organizations against them.

The widespread use of APIs in numerous industries makes them a prime target for attackers. Organizations need to understand and address the vulnerabilities identified in the OWASP Top 10 to protect their vital assets and maintain user trust.

This whitepaper examines each of the OWASP API Security Top 10 threats in detail, showcasing examples and illustrating how Salt Security's solutions can help organizations stay ahead of emerging threats.

API1:2023 Broken Object Level Authorization

Description

APIs typically provide endpoints that manage object identifiers, which creates a broad attack surface. Object-level authorization, usually enforced in the code, ensures a user's access to particular objects. Today's applications incorporate intricate and widespread access control and authorization mechanisms. Despite having the correct authorization frameworks in place, developers may neglect to implement checks before accessing objects. Attackers can take advantage of exposed API endpoints by altering object IDs in API requests. Such vulnerabilities are prevalent in API-driven applications, as servers often depend on client-provided parameters, such as object IDs, to establish access rights.

Legitimate – *userId* matches in the query parameter and request

```
Request:
GET /v1/customers/15981?userId=207939055
HTTP/1.1

Authorization: Bearer gwwh1Y4epjv9Y

Cookie: _ga=GA1.3.630674023.1502871544;
_gid=GA1.2.1579405782.1502871544;userId=207939
055Host: payments-api.dnssf.com
X-Forwarded-For: 54.183.50.90

Response:
200 OK

{
     userId: 207939055,
     firstName: ”John”,
     lastName: “Smith”,
     email: “john.smith@acme.com”,
     phoneNumber: “+1650123123”
}
```

Attack - Attacker changes the *userId* in the query parameter

```
Request:
GET /v1/customers/15981?userId=207938044
HTTP/1.1

Authorization: Bearer gwwh1Y4epjv9Y

Cookie: _ga=GA1.3.630674023.1502871544;
_gid=GA1.2.1579405782.1502871544;userId=207939
055Host: payments-api.dnssf.com
X-Forwarded-For: 54.183.50.90

Response:
200 OK

{
    userId: 207938044,
     firstName: ”David”,
     lastName: “Miller”,
     email: “david.miller@example.com”,
     phoneNumber: “+1912456456”
}
```

Potential Impact
Broken Object Level Authorization (BOLA) can lead to data exfiltration, unauthorized data viewing, modification, or destruction. In severe cases, BOLA can result in full account takeover, such as an attacker compromising a password reset flow to reset credentials for an unauthorized account.

Example
A common BOLA scenario involves an attacker changing the userId in a query parameter to access unauthorized data. If userIds are sequential, the attacker can enumerate the parameter to scrape large amounts of data, especially if rate limits are not enforced.

Why Existing Tools Fail
Traditional security tools like WAFs and API gateways often miss BOLA attacks because they lack API context and do not baseline normal API usage. They cannot identify unknown attack patterns like BOLA, which do not rely on predictable patterns like code injection.

How Salt Prevents BOLA Attacks
Salt Security's solution learns the business logic of an API and detects when one user tries to access another user's data without authorization. By analyzing large amounts of API traffic, Salt establishes a baseline of normal usage and identifies anomalies like manipulating userId in requests.

API2:2023 Broken User Authentication

Description
API authentication is a complicated topic, frequently misunderstood and prone to implementation errors. Authentication methods are appealing targets for hackers, particularly when they are publicly accessible. Sophisticated attacks on authentication encompass brute-force attempts, credential stuffing, and credential cracking.

Broken user authentication arises from two main issues: insufficient protection of authentication endpoints and incorrect implementation of authentication systems. Technical issues that lead to broken authentication include weak passwords, short password histories, a lack of account lockout policies, and insecure management of authentication data.

Potential Impact
Attackers may take advantage of weak authentication mechanisms to hijack user accounts, access confidential information, and execute unauthorized transactions. In scenarios involving machine-to-machine communication, breaching authentication can expose all data linked to the machine's identity. Comparable assaults can also focus on workload authentication as well as server-side API metadata services within cloud-native settings.

Example
Common attacks targeting broken authentication include API enumeration and brute-forcing, often involving high volumes of API requests with minor changes. For instance, attackers can exploit password recovery mechanisms by enumerating password reset tokens if rate limiting is not implemented.

Why Existing Tools Fail
Conventional security tools, such as WAFs and API gateways, frequently do not provide detailed enforcement of

authentication. Instead, they primarily check for session identifiers or authentication tokens. Although API gateways can apply authentication policies, they often miss the context needed to ascertain the right authentication methods for particular API scenarios. Moreover, these traditional tools are unable to monitor attack traffic over time, hindering the detection of sophisticated attacks like credential stuffing.

How Salt Prevents User Authentication Attacks
Salt Security's solution profiles typical authentication sequences for each API flow, enabling it to detect anomalies like missing credentials or out-of-sequence authentication calls. By analyzing production API traffic, Salt establishes a baseline and identifies abnormal behavior, mitigating advanced authentication attacks.

API3:2023 Broken Object Property Level Authorization

Description
In 2023, this new category merges the earlier "Excessive Data Exposure" and "Mass Assignment" categories. It highlights the necessity for detailed authorization that applies not only at the object level but also at the property level within those objects. APIs frequently present objects with differing access levels across various properties, and it is essential to enforce these distinctions.

Potential Impact
Inadequate authorization for object property access may result in unauthorized exposure, alteration, or data removal. Malicious actors can exploit these weaknesses to access sensitive information or alter data for harmful ends.

Example
A common scenario involves an API endpoint that allows users to update certain properties of an object but lacks proper validation for sensitive properties. An attacker could manipulate the request to modify these sensitive properties without authorization.

Why Existing Tools Fail
Traditional security tools often focus on object-level authorization but lack the granularity to enforce property-level access controls. They may not be able to distinguish between authorized and unauthorized access to specific properties within an object.

How Salt Prevents Broken Object Property Level Authorization Attacks
Salt Security's solution examines API traffic to comprehend the connections between objects and their attributes, enabling the identification and implementation of precise access controls. This safeguards against attackers leveraging vulnerabilities tied to unauthorized access or alterations of object properties.

API4:2023 Unrestricted Resource Consumption

Description
API requests utilize various resources, including network bandwidth, CPU, memory, and storage. Since APIs often lack limits on the size or quantity of resources clients can request, this can result in Denial of Service (DoS) attacks and facilitate brute force or enumeration attacks on authentication and data-fetching APIs.

Legitimate – max_return and page_size request attributes are normal

```
POST
/example/api/v1/provision/user/search
HTTP/1.1
User-Agent: AHC/1.0
Connection: keep-alive
Accept: */*
Content-Type: application/json;
charset=UTF-8
Content-Length: 131
X-Forwarded-For: 10.93.23.4

{
  "search_filter":
"user_id=exampleId_100",
  "max_return": "250",
  "page_size": "250",
  "return_attributes": [

  ]
}
```

Attack – Attackers modify the request to return an abnormally high response size

```
POST
/example/api/v1/provision/user/search
HTTP/1.1
User-Agent: AHC/1.0
Connection: keep-alive
Accept: */*
Content-Type: application/json;
charset=UTF-8
Content-Length: 131
X-Forwarded-For: 10.93.23.4

{
  "search_filter":
"user_id=exampleId_100",
  "max_return": “20000",
  "page_size": "20000",
  "return_attributes": [

  ]
}
```

Potential Impact

Insufficient resource limits enable attackers to create single API calls that can inundate an application, affecting its performance and responsiveness. This vulnerability can result in application-level DoS attacks, which may expose the system to authentication threats and data breaches. Furthermore, the absence of rate limiting lets attackers flood the system with numerous API requests, exhausting resources, engaging in credential brute-forcing, and extracting significant amounts of data.

Example

An attacker can manipulate parameters like max_return and page_size in a search query to request an excessive number of items, potentially slowing down or crashing the application for all users.

Why Existing Tools Fail

Traditional security tools like WAFs and API gateways often offer basic or static rate limiting, which is difficult to enforce at scale. These tools lack the context to determine normal API behavior, making it challenging to set appropriate limits without impacting legitimate functionality. They may also only cover ingress traffic, leaving egress traffic vulnerable.

How Salt Prevents Unrestricted Resource Consumption

Salt identifies API calls and parameter values that fall outside normal usage patterns. By analyzing API traffic and establishing baselines, Salt can detect and block requests that deviate from expected behavior, preventing resource abuse and rate limiting attacks

API5:2023 Broken Function Level Authorization

Description
Authorization flaws often result from improperly implemented or misconfigured authorization mechanisms. Modern applications with complex roles, groups, and user hierarchies, coupled with distributed architectures and cloud-native designs, make implementing adequate authorization challenging. Broken Function Level Authorization (BFLA) is similar to BOLA, but it targets API functions instead of objects. Attackers exploit BFLA to escalate privileges horizontally or vertically.

Legitimate – POST method is correctly requested

```
POST
/example/api/v1/provision/user/search
HTTP/1.1
User-Agent: AHC/1.0
Connection: keep-alive
Accept: */*
Content-Type: application/json;
charset=UTF-8
Content-Length: 131
X-Forwarded-For: 10.93.23.4

{
  "search_filter":
"user_id=exampleId_100",
  "max_return": "250",
  "page_size": "250",
  "return_attributes": [

  ]
}
```

Attack – Request is modified to send a DELETE method

```
DELETE
/example/api/v1/provision/user/search
HTTP/1.1
User-Agent: AHC/1.0
Connection: keep-alive
Accept: */*
Content-Type: application/json;
charset=UTF-8
Content-Length: 131
X-Forwarded-For: 10.93.23.4

{
  "search_filter":
"user_id=exampleId_100",
  "max_return": “250",
  "page_size": “250",
  "return_attributes": [

  ]
}
```

Potential Impact
Attackers exploiting BFLA vulnerabilities can access unauthorized resources, take over user accounts, create or delete accounts, and gain administrative access.

Example
An attacker can change an HTTP method from GET to DELETE to delete an account, or modify a parameter to escalate privileges.

Why Existing Tools Fail
Traditional security tools lack context and cannot determine if an attacker should be able to send a specific HTTP method or access certain API functions. They may offer basic message filtering, but these approaches can be too restrictive and difficult to manage at scale.

How Salt Prevents BFLA Attacks
Salt continuously baselines typical HTTP access patterns per API endpoint and user, enabling it to identify calls with unexpected parameters or HTTP methods. This allows Salt to detect and prevent attackers from accessing unauthorized functionality or administrative capabilities.

API6:2023 Unrestricted Access to Sensitive Business Flows

Description:
This new category for 2023 focuses on protecting sensitive business flows that are exposed through APIs. Attackers can exploit these flows to perform unauthorized actions or access sensitive data if they are not adequately protected. Business logic flaws in APIs can allow attackers to manipulate legitimate functionality for malicious purposes.

Potential Impact:
Unrestricted access to sensitive business flows can lead to financial loss, data breaches, denial of service, and reputational damage. Attackers can exploit these vulnerabilities to perform actions such as transferring funds, creating fake accounts, or manipulating sensitive data.

Example:
An attacker could manipulate an API endpoint that manages user account balances to transfer funds from one account to another without authorization.

Why Existing Tools Fail:
Traditional security tools often focus on technical vulnerabilities but may not be able to identify and protect against business logic flaws in APIs. These tools typically lack the context to understand the business logic and identify malicious manipulation of legitimate functionality.

How Salt Prevents Unrestricted Access to Sensitive Business Flows:
Salt Security's solution examines API traffic to comprehend business logic and pinpoint sensitive workflows. It can subsequently detect and obstruct attacks aimed at manipulating these workflows for harmful objectives. This safeguards against adversaries taking advantage of business logic vulnerabilities to carry out unauthorized actions or obtain sensitive information.

API7:2023 Server-Side Request Forgery (SSRF)

Description
The 2023 category details SSRF vulnerabilities, which happen when an API retrieves a remote resource without validating the user-provided URL. This flaw can enable attackers to manipulate the application into sending specially crafted requests to unanticipated targets, possibly circumventing firewalls and VPNs. Current development techniques, including webhooks and custom SSO, frequently require accessing external resources guided by user input, heightening the risk of SSRF.

Potential Impact
Attackers can exploit SSRF to access internal resources, scan internal networks, and even execute arbitrary code on the server.

Example
An attacker can manipulate a URL parameter in an API request to point to an internal server or service, potentially gaining access to sensitive information or disrupting internal systems.

Why Existing Tools Fail
Traditional security tools often struggle to detect and prevent SSRF attacks because they may not be able to distinguish between legitimate and malicious requests to external resources.

How Salt Prevents SSRF Attacks
Salt Security's solution analyzes API traffic to identify and block requests to unexpected or unauthorized destinations, preventing SSRF attacks. It can also validate user-supplied URLs to ensure they comply with security policies.

API8:2023 Security Misconfiguration

Description
Security Misconfiguration encompasses a wide range of errors that can jeopardize API security. This includes insecure default settings, unprotected cloud storage, incorrect HTTP headers, unused HTTP methods, overly permissive CORS configurations, and overly detailed error messages.

Legitimate – Client sends a legitimate request

```
GET /api/v2/network/connections/593065
HTTP/1.1
Accept: application/json, text/plain, */*
Accept-Encoding: gzip

HTTP/1.1 200 OK
{
"status":”success",
}
```

Attack – Attackers modify the connectionId resulting in a detailed exception error

```
GET /api/v2/network/connections/5930aaaaa
HTTP/1.1
Accept: application/json, text/plain, */*
Accept-Encoding: gzip

HTTP/1.1 500 Server Error
{
"status":"failure",
"statusMessage":"An error occurred while
validating input: validation error: unexpected
content \"593065d1\"
({com.tibco.xml.validation}COMPLEX_E_UNEXPECTE
D_CONTENT) at
/{http://www.tibco.com/namespaces/tnt/plugins/
json}ActivityOutputClass[1]/searchSvcReqsByRep
Req[1]/search[1]/status[1]/aaaa[1]\ncom.tibco.
xml.validation.exception.UnexpectedElementExce
ption: unexpected content \"aaaa\"&#xD;\n\tat
com.tibco.xml.validation.state.a.a.a(CMElement
ValidationState.java:476)&#xD;\n\tat
com.tibco.xml.validation.state.a.a.a(CMElement
ValidationState.java:270)&#xD;\n\tat
com.tibco.xml.validation.state.driver.Validati
onJazz.c(ValidationJazz.java:993)&#xD;\n\tat
com.tibco.xml.validation.state.driver.Validati
onJazz.b(ValidationJazz.java:898)&#xD;\n\tat
….
```

Potential Impact
During reconnaissance, attackers may use security misconfigurations to gather information about application and API components. Comprehensive error messages can inadvertently reveal sensitive user data and system details, providing attackers with insights into potential vulnerabilities. Additionally, these misconfigurations can be leveraged to launch attacks against APIs, such as circumventing authentication because of improperly configured access controls mechanisms.

Example
An attacker can modify an API request to trigger an error response that reveals sensitive information about the application environment, such as software versions and code snippets.

Why Existing Tools Fail
Traditional security tools may not identify all security misconfigurations, especially those that are not related to known vulnerabilities or predictable attack patterns. They may also lack the context to determine if an error response contains sensitive information.

How Salt Prevents Security Misconfiguration Vulnerabilities
Salt identifies misconfigurations and security vulnerabilities in APIs, recommending remediation when attacks are attempted. It examines API activity to set baselines and pinpoint excessive or sensitive information in error messages. Additionally, Salt can detect early reconnaissance actions by attackers searching for security gaps.

API9:2023 Improper Inventory Management

Description
Keeping a thorough and current API inventory with precise documentation is essential for recognizing potential security threats. An outdated or incomplete inventory may create unidentified gaps in the API attack surface, complicating the process of identifying and retiring obsolete, vulnerable APIs. Additionally, inaccurate documentation can lead to unforeseen exposure of sensitive information and obstruct efforts to remediate vulnerabilities.

Potential Impact
Attackers can exploit unknown or undocumented APIs (shadow APIs) and forgotten APIs (zombie APIs) to gain unauthorized access to sensitive data or even gain full server access.

Example
Research by Salt Security has shown that there can be a significant discrepancy between manually created API documentation and actual API deployments. This can lead to shadow API endpoints, shadow parameters, and parameter definition discrepancies, leaving APIs vulnerable to attacks.

Why Existing Tools Fail
Traditional security tools cannot continuously discover and monitor APIs for changes. They rely on API schema definitions, which may be missing or inaccurate, leading to an incomplete view of the API environment.

How Salt Prevents Improper Inventory Management
Salt reviews all API traffic and consistently identifies APIs, encompassing all host addresses, API endpoints, HTTP methods, parameters, and data types. This guarantees a current inventory of the API landscape and precise documentation, even as APIs develop over time.

API10:2023 Unsafe Consumption of APIs

Description
This new category focuses on the risks associated with consuming third-party APIs. It highlights the importance of validating and sanitizing data received from external APIs before using it.

Potential Impact
Unsafe consumption of APIs can lead to data breaches, injection attacks, and other vulnerabilities if data from external APIs is not properly handled.

Example
An attacker could compromise a third-party API and inject malicious data into the responses, which could then be used to exploit vulnerabilities in the consuming application.

Why Existing Tools Fail
Traditional security tools often focus on protecting against attacks originating from external sources but may not adequately address the risks associated with consuming third-party APIs.

How Salt Prevents Unsafe Consumption of APIs
Salt Security's solution examines API traffic to verify and validate data from external APIs, ensuring its safety before being utilized by the application. This proactive approach blocks attackers from exploiting weaknesses associated with insecure API usage.

Conclusion

Protecting APIs from the OWASP API Security Top 10 risks requires a modern security approach. Traditional methods and tools are inadequate for addressing the evolving threat landscape. Salt Security's API Protection Platform provides a comprehensive solution for API security by leveraging big data, AI, and machine learning to detect, protect, and remediate vulnerabilities. By continuously monitoring API traffic, Salt offers detailed context, real-time analysis, and continuous insights to help organizations defend against API attacks and enable confident innovation.


SALT